Information Processing Letters 39 (1991) 343–350
North-Holland

27 September 1991

The causal ordering abstraction and a simple way to implement it

Michel Raynal
IRISA, Campus de Beaulieu, F-35042 Rennes Cédex, France

André Schiper
Ecole Polytechnique Fédérale, Département d'Informatique, CH-1015 Lausanne, France

Sam Toueg
Department of Computer Science, Cornell University, Ithaca, NY 94853, USA

Communicated by L. Kott
Received 11 May 1990
Revised 3 April 1991

Abstract

Raynal, M., A. Schiper and S. Toueg, The causal ordering abstraction and a simple way to implement it, Information Processing Letters 39 (1991) 343–350.

Control in distributed systems is mainly introduced to reduce nondeterminism. This nondeterminism is due on the one hand to the asynchronous execution of the processes located on the various sites of the system, and on the other hand to the asynchronous nature of the communication channels. In order to limit the asynchronism due to the communication channels, a new message ordering relation, known as causal ordering, has been introduced by Birman and Joseph. After giving some examples of this causal ordering, we propose a simple algorithm to implement it. This algorithm is based on message sequence numbering. A proof of the correctness of the algorithm is also given.

Keywords: Distributed computing, distributed algorithm, causal ordering, network protocol

1. Controlling nondeterminism

Controlling nondeterminism is one of the essential tasks in computer systems. In centralized systems the nondeterminism is caused by the interrupts. In distributed systems the nondeterminism is also due to the asynchronous execution of the parallel processes together with the asynchronous nature of the communication channels. Moreover, such systems can also be affected by the nondeterministic occurrences of component failures.

In order to reduce the asynchronism of the processes and to solve the related problems (e.g. resource allocation problem) one usually introduces synchronization mechanisms [14,12]. In order to reduce the asynchronism of the communication channels it is possible to build network synchronizers which force processes and channels to progress in synchronized steps [1,7]. This corresponds to defining a virtual distributed machine in which the undesirable behaviour due to nondeterminism has been suppressed. The same is true for communication protocols, whose objective is

effectively to mask nonreliability channels: communication protocols define a "reliable channel" abstraction which is implemented on top of a nonreliable channel. The alternate bit protocol [2] or Stenning's protocol [16] are such examples: both build a channel without message loss, duplication or desequencing on top of a channel that can lose or duplicate messages, or even, in the case of Stenning's protocol, that can desequence them. The reliable channel abstraction, like the synchronization mechanisms, eliminates the undesirable behaviours.

In this paper we consider a reliable distributed system, and are concerned by the realization of a communication scheme that eliminates a particular undesirable behaviour. More precisely, we are concerned with the abstraction called "causal ordering", as proposed by the Isis system [3]. The paper is organized in the following way. In Section 2 we define causal ordering and show its usefulness. In Section 3 we show an easy and natural way to implement causal ordering (which differs from the Isis implementation), and give in Section 4 a proof of its correctness.

2. Causal ordering

2.1. Definition

Causal ordering of events in a distributed system (an event corresponding to the emission of a message, the reception of a message, or an internal action) is based on the well-known "happened before" relation, noted $\rightarrow$ [9]. If A and B are two events, then $A \rightarrow B$ if and only if one of the following conditions is true:

(i) A and B are two events occurring on the same site, A occurs before B;

(ii) A is the emission of a message, and B corresponds to the reception of the same message;

(iii) there exists an event C such that $A \rightarrow C$ and $C \rightarrow B$.

Causal ordering is concerned with the order in which two messages M_1 and M_2, such that SEND(M_1) $\rightarrow$ SEND(M_2), are delivered. Let us introduce the notation DELIVERY(M) to indicate the event corresponding to the delivery of M to its destination process, and consider two events SEND(M_1) and SEND(M_2). Causal ordering is respected if:

> if SEND(M_1) $\rightarrow$ SEND(M_2), and M_1, M_2 have the same destination, then DELIVERY(M_1) $\rightarrow$ DELIVERY(M_2)

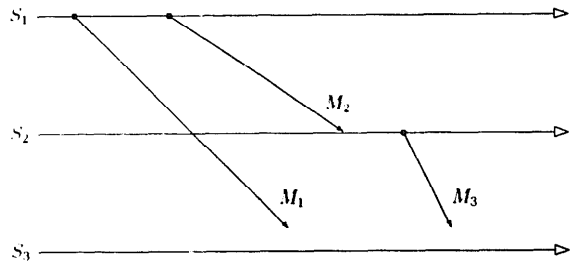

Fig. 1. Causal ordering is respected if M_1 is delivered before M_3.

In other words, in case of a "happened before" relation between two emissions to a same destination, there exists a "happened before" relation between the delivery of the messages. For example, in Fig. 1, causal ordering is respected if message M_1 is delivered on site S_3 before message M_3.

When sending messages using traditional transport protocols, causal ordering might clearly be violated. Implementing causal ordering means adding a protocol to the original system such that causal ordering is never violated in the system equipped with the superimposed protocol. To avoid confusion between both systems, we will hereafter consistently use the expression message "reception" to denote a message received by the original system, and message "delivery" to denote a message received by the system equipped with the causal ordering protocol. An analogy can be drawn here between FIFO channels and a system implementing causal ordering. A FIFO channel reduces the nondeterminism of a point to point communication. A system implementing causal ordering globally reduces the nondeterminism of the communications.

2.2. Examples of the usefulness of causal ordering

Management of a replicated data

Consider a data X replicated on various sites. In order to ensure mutual consistency of the vari-

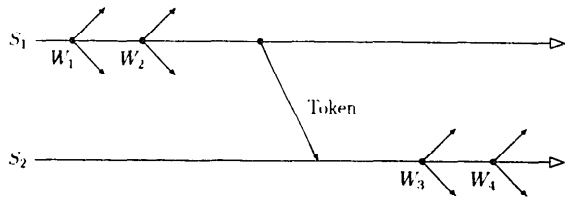

Fig. 2. Management of a replicated data.

ous copies x_1, x_2,..., the updates to these copies must take place in the same order. In particular, this introduces the need for mutual exclusion on the updates done by two different sites. This can be solved using a token; when in possession of the token, a site can update his copy of X and broadcast the update to all other sites having a copy of X (Fig. 2). Causal ordering ensures that all sites will receive all the updates in the same order (W_i will not be delivered before W_j, for $i < j$), which ensures mutual consistency of the copies [4,8]. The token ensures total ordering of the updates on X and causal ordering ensures that all the copies x_i are updated in this same order.

Observing a distributed system

Causal ordering can also be used to provide consistent observations of a distributed system. Consider a number of sites monitoring events occurring locally, and suppose that all this information is of interest to an observer site OS. If the monitoring information collected by the sites is sent to OS respecting causal ordering, the information will be received in a coherent order (i.e. in an order not violating the events precedence order). For example, in Fig. 1 the observer S_3 receives probe messages M_1, then M_3, that is to say in a consistent observation order.

Resource allocation

Consider the mutual exclusion problem, or more generally a resource allocation problem. This can be solved by a resource allocator residing on a site, which receives allocation requests in a FIFO queue. With this scheme, the requests are honoured in the reception order. This might however be considered unsatisfactory in some cases: it might be desirable to honour requests not in their reception order, but in their emission order, i.e. if two requests R_1 and R_2 are such that $R_1 \rightarrow R_2$, then R_1 should be honoured before R_2. This is again a causal ordering problem (consider in Fig. 1 that M_1 carries request R_1 and M_3 carries request R_2). Note that this example, in the particular case of mutual exclusion, was used by Lamport in his paper introducing logical clocks [9]. Lamport's solution was to totally order the requests, which is not necessary if messages are causally ordered.

3. Implementation of causal ordering

3.1. Related work

The Isis implementation of causal ordering [3] is straightforward: a message carries with it all the history of the communications that have preceded. Consider again Fig. 1. Message M_3, sent by S_2 to S_3, carries the following information about its past: message M_1, with the information that it was sent to S_3, and message M_2 with the information that it was sent to S_2. When receiving M_3, site S_3 thus also receives a copy of M_1: if not already delivered, M_1 will be delivered at that time, i.e. before M_3. To prevent unbounded growth of the information added to a message, a mechanism must be added to inform of messages that have been received and delivered: these messages need not be sent any more. This mechanism is however rather complex. Anyhow, the size of the information added to messages is unbounded.

Another implementation of causal ordering is given in [15]. In this implementation, control information (and not messages as in Isis) is added to the messages. This control information allows the destination site of a message M to know if there are messages that have to be delivered before M, in order to respect causal ordering: if this is the case, M is not immediately delivered. The control information is composed of a bounded number of pairs (destination site, vector time), where "vector time" is a time defining a partial ordering of the events of a distributed system [5,11]. The implementation proposed in this paper has some similarities with this implementation. We do not use vector times, and thus this algorithm is much easier to understand.

3.2. A simple implementation

Consider that we have an original noncausal (i.e. where causal ordering might be violated) system NC. Our objective is to define a protocol in the NC system in order to build a new system C in which causal ordering is never violated. To implement the protocol we give every site in NC control information representing the site's perception of the system state (more precisely, the site's perception of the communications). This information will allow each site to decide when a received message can be delivered to the C system. Moreover, information added to every message will allow the receiving site in NC, when a message has been delivered, to update its perception of the system state.

Local information of a site

Every site in the NC system manages the following two variables (where n is equal to the number of sites in the system):

DELIV: array[1..n] of integer;

(* initially DELIV[i] = 0 for all i *)

SENT: array[1..n, 1..n] of integer;

(* initially SENT[i, j] = 0 for all i, j *)

We will use the notation DELIV_i and SENT_i to refer to the variables managed by site S_i. So on site S_i, the variable $\text{DELIV}_i[j]$ represents the number of messages sent from S_j and delivered to S_i; the variable $\text{SENT}_i[k, l]$ represents the knowledge of S_i of the number of messages sent (but not necessarily delivered) from S_k to S_l (more precisely S_i has been informed that at least $\text{SENT}_i[k, l]$ messages were sent from S_k to S_l).

To illustrate these definitions consider Fig. 3, and more precisely $\text{SENT}_j[3, 1]$ which is equal to 1, meaning that site j knows that a message (here M_1) has been sent from site $k = 3$ to site $i = 1$. As we will see, this information has been sent together with M_2.

Behaviour of a site

The behaviour of a site S_i is expressed by the following two rules, governing the emission and the reception of a message in the NC system.

(i) *Emission of a message M from S_i to S_j:*

send(M, SENT_i) to S_j;

$\text{SENT}_i[i, j] := \text{SENT}_i[i, j] + 1$;

Thus M is sent together with a copy of SENT_i, i.e. with site S_i's view of the system state.

(ii) *Reception of (M, ST_M) sent from S_j to S_i, where ST_M represents the control information* "SENT" *carried by the message M* (see Fig. 3):

wait(for all k, $\text{DELIV}_i[k] \geqslant \text{ST}_M[k, i]$);

delivery of M to the C system;

$\text{DELIV}_i[j] := \text{DELIV}_i[j] + 1$;

$\text{SENT}_i[j, i] := \text{SENT}_i[j, i] + 1$;

for all k, l: $\text{SENT}_i[k, l]$

$:= \max(\text{SENT}_i[k, l], \text{ST}_M[k, l])$;

Thus a message M sent by S_j can be delivered to S_i only if for all k, $\text{DELIV}_i[k] \geqslant \text{ST}_M[k, i]$, which expresses that all causally preceding messages, whose existence is revealed by ST_M, have been delivered. Consider for example message M_4 in Fig. 3. $\text{ST}_{M_4}[3, 1] = 1$ which means that M_4 can be delivered to site 1 when $\text{DELIV}_1[3] \geqslant 1$. This is true as soon as M_4 is delivered. Note on this particular example that $\text{DELIV}_1[3] > \text{ST}_{M_4}[3, 1]$ when M_4 is received on site 1. This situation is due to the existence of M_3, whose emission does not causally precede the emission of M_4: $\text{SEND}(M_3) \rightarrow \text{SEND}(M_4)$. The two events $\text{SEND}(M_3)$ and $\text{SEND}(M_4)$ are said to be concurrent.

Once a message has been delivered, the site's perception of the system state is updated in an obvious way.

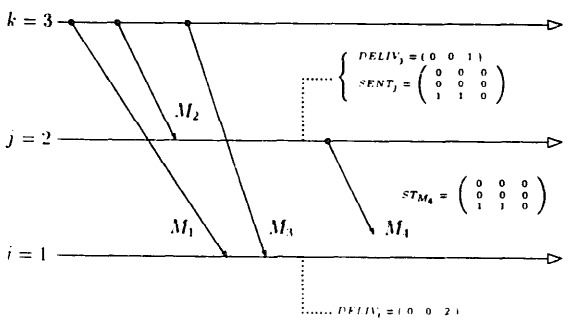

Fig. 3. Illustration of the data structures.

4. Proof of correctness

The correctness of the above algorithm will be proved in the usual two steps; safety and liveness. However, before developing these proofs we need some intermediate results that will be given in Section 4.1. We will then in Section 4.2 prove the safety of our protocol, which corresponds to proving that causal ordering is never violated. Finally in Section 4.3 we will prove the liveness of the algorithm, which corresponds to proving that every message will eventually be delivered. Concerning the underlying NC system on which the causal ordering is implemented, we need the assumption that every message sent is received, i.e. no messages are lost in NC, and the NC system is live.

4.1. Preliminary results

In order to prove the safety and liveness of our protocol, we need some preliminary results.

Proposition 1. *Consider a message M sent by S_j to S_i such that $\mathrm{ST}_M[k, i] > 0$ (including $k = j$), and let $\mathrm{ST}_M[k, i] = x$. Then M_x, the xth message sent by S_k to S_i, is such that* $\mathrm{SEND}(M_x) \rightarrow \mathrm{SEND}(M)$.

Proof. (i) $k = j$. From $\mathrm{ST}_M[j, i] = x$ and the emission protocol, $\mathrm{SENT}_j[j, i] = x$ at the time M is sent. Moreover, it follows directly from the protocol that $\mathrm{SENT}_j[j, i]$ is never incremented on reception of a message by S_j. Thus at the time M is sent, the emission protocol has been executed x times by S_j, i.e. M_x has been sent before M, and $\mathrm{SEND}(M_x) \rightarrow \mathrm{SEND}(M)$.

(ii) $k \neq j$. The proof will consist in building a causal chain $\mathrm{SEND}(M_x) \rightarrow \cdots \rightarrow \mathrm{SEND}(M)$. Let X_j be the first message sent by S_j such that $\mathrm{ST}_{X_j}[k, i] = x$ (Fig. 4), i.e. $\mathrm{SEND}_j[k, i] = x$ when X_j was sent (X_j might be M). Thus before the emissions of X_j, S_j must have received a message $M_{j'}$ from $S_{j'}$ with $\mathrm{ST}_{M_{j'}}[k, i] = x$ (see the last step of the delivery protocol), and $\mathrm{SEND}(M_{j'}) \rightarrow \mathrm{SEND}(M)$.

Because $\mathrm{ST}_{M_{j'}}[k, i] = \mathrm{ST}_M[k, i]$, the same construction applied to M can be applied to $M_{j'}$, and so on. Each iteration of this construction introduces a new site $S_j(s)$ different from all previous sites, because $\rightarrow$ defines an order [9]: if $S_{j(s1)} = S_{j(s2)}$ for $s1 \neq s2$, we would have a cycle in the causal chain built.

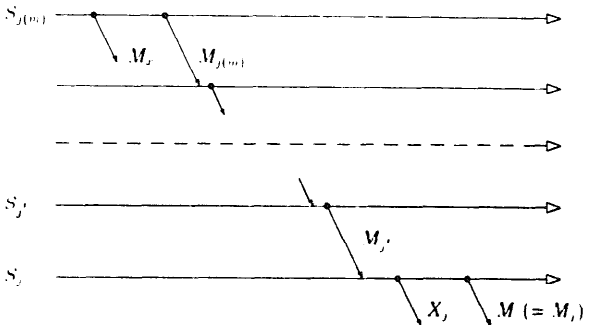

Fig. 4. Construction of the causal chain $\mathrm{SEND}(M_x) \rightarrow \cdots \rightarrow \mathrm{SEND}(M)$.

As the number of sites is finite, we finally end up with $S_{j(m)} = S_k$, $M_{j(m)}$ sent by $S_{j(m)}$, and $\mathrm{ST}_{M_{j(m)}}[k, i] = x$, which has been treated under (i). Thus the causal chain $\mathrm{SEND}(M_x) \rightarrow \mathrm{SEND}(M_{j(m)}) \rightarrow \cdots \rightarrow \mathrm{SEND}(M_{j'}) \rightarrow \mathrm{SEND}(M)$ has been built. □

The two following lemmas are intermediate results needed to prove Proposition 4.

Lemma 2. *Let M_x be the xth message sent by S_j to S_i ($x > 0$), and consider a message M such that* $\mathrm{SEND}(M_x) \rightarrow \mathrm{SEND}(M)$. *Then* $\mathrm{ST}_M[j, i] \geq x$.

Proof (sketch). From the emission protocol, it follows immediately that $\mathrm{ST}_{M_x}[i, j] = x - 1$. From this, the lemma can easily be proven by considering the emission and reception protocol applied to any causal chain connecting $\mathrm{SEND}(M_x)$ to $\mathrm{SEND}(M)$. □

Lemma 3. *Consider a message M and let $\mathrm{ST}_M[j, i] = x$. Then M_{x+1}, the $(x+1)$st message sent by S_j to S_i is such that* $\mathrm{SEND}(M_{x+1}) \rightarrow \mathrm{SEND}(M)$.

Proof. Suppose that $\mathrm{SEND}(M_{x+1}) \rightarrow \mathrm{SEND}(M)$. By Lemma 2 we have $\mathrm{ST}_M[j, i] \geq x + 1$, which shows the contradiction. □

Proposition 4. *Consider M sent by S_j to S_i, and another site S_k (including $k = j$). Then $\mathrm{ST}_M[k, i]$ is equal to the number of messages M' sent by S_k to S_i such that* $\mathrm{SEND}(M') \rightarrow \mathrm{SEND}(M)$.

Proof. Let m be the number of messages M' sent by S_k to S_i such that SEND(M') → SEND(M), and let $ST_M[k, i] = x$.

(i) By Proposition 1, the xth message sent from S_k to S_i is such that SEND(M_x) → SEND(M). Thus $m \geqslant x$.

(ii) Consider M_{x+1}, the $(x+1)$st message sent from S_k to S_i. By Lemma 3, SEND(M_{x+1}) → SEND(M). Thus $m \leqslant x$.

From (i) and (ii) we conclude that $m = x$. □

Corollary 5. *Consider a message M_1 sent from S_j to S_i, and a message M_2 such that* SEND(M_1) → SEND(M_2). *Then* $ST_{M_1}[j, i] < ST_{M_2}[j, i]$.

Proof. By Proposition 4, $ST_{M_1}[j, i]$ is equal to the number of messages M' sent from S_j to S_i such that SEND(M') → SEND(M_1). But we have SEND(M') → SEND(M_1) → SEND(M_2). Thus because M_1 is also sent from S_j to S_i, by Proposition 4, $ST_{M_2}[j, i]$ is at least equal to $ST_{M_1}[j, i] + 1$. This allows to conclude that $ST_{M_1}[j, i] < ST_{M_2}[j, i]$. □

Proposition 6. *Consider a message M sent from S_j to S_i. If neither M, nor any message M' such that* SEND(M) → SEND(M') *has been delivered to S_i, then* $DELIV_i[j] \leqslant ST_M[j, i]$.

Proof. We have only to consider the messages M' sent by S_j, as only the delivery of these messages can increase $DELIV_i[j]$. By Proposition 4, $ST_M[j, i]$ is equal to the number of messages M'' sent by S_j to S_i such that SEND(M'') → SEND(M). By hypothesis, only these messages M'' could have been delivered to S_i. Thus $DELIV_i[j] \leqslant ST_M[j, i]$. □

Finally, Proposition 7 will only be used to prove the liveness of our protocol. For this reason we can, in Proposition 7, make the assumption that our protocol is safe, i.e. if SEND(M_1) → SEND(M_2) and M_1, M_2 are sent to the same destination, then M_2 cannot be delivered before M_1 (safety will be proved in Section 4.2).

Proposition 7. *Suppose that our protocol is safe, and consider M_x, the xth message sent from S_j to S_i ($x > 0$). Then if* $DELIV_i[j] < x$, *M_x has not been delivered to S_i.*

Proof. $DELIV_i[j]$ is incremented each time a message from S_j is delivered. Thus if $DELIV_i[j] < x$, less than x messages from S_j have been delivered to S_i. As the protocol is supposed to be safe, at most the $(x-1)$ first messages from S_j could have been delivered to S_i. Thus M_x has not been delivered to S_i. □

4.2. Safety

Safety corresponds to proving that causal ordering is never violated, i.e. if SEND(M_1) → SEND(M_2) and M_1, M_2 are sent to the same destination, then M_2 cannot be delivered before M_1. This can be reformulated in the following way. Consider a message M sent from S_j to S_i, and let M_n denote the nth message delivered to S_i. Then for all n, as long as M is not delivered, SEND(M) → SEND(M_n) cannot be true. This formulation is adapted to a proof by induction.

(i) *Base step* ($n = 1$). We are going to suppose that M_1, the first message delivered to S_i, is such that SEND(M) → SEND(M_1), and show that we end up with a contradiction. By Proposition 4, as M is sent from S_j to S_i, $ST_{M_1}[j, i] \geqslant 1$. Initially $DELIV_i[j] = 0$, so when M_1 is received by S_i, $DELIV_i[j] < ST_{M_1}[j, i]$: the delivery condition (Section 3.2) is not satisfied, i.e. M_1 cannot be delivered, which shows the contradiction.

(ii) *Induction step.* By induction hypothesis, as long as M is not delivered, for all $p \leqslant n$, SEND(M) → SEND(M_p) cannot be true. We are going to prove that the same holds for $n+1$. Let us again suppose that SEND(M) → SEND(M_{n+1}) and show the contradiction.

Before M_{n+1} is delivered by S_i, we have by Proposition 6: $DELIV_i[j] \leqslant ST_M[j, i]$. Furthermore by Corollary 5 we have $ST_M[j, i] < ST_{M_{n+1}}[j, i]$, i.e. $DELIV_i[j] < ST_{M_{n+1}}[j, i]$ before the reception of M_{n+1} by S_i. Thus the delivery condition is not satisfied (Section 3.2), i.e. M_{n+1} cannot be delivered. This again shows the contradiction.

4.3. Liveness

As already said, we will prove here that every message is eventually delivered. Consider two messages M', M'' and define $M' < M''$ iff $\text{SEND}(M') \rightarrow \text{SEND}(M'')$. We suppose that a message is delivered as soon as it is received and its delivery condition becomes true. So a message cannot be delivered either if it is not received, or if it is received and its delivery condition is false. Consider then a site S_i and all the messages sent to it that cannot be delivered. Using the $<$ relation, let M_s be one of the smallest of these messages (there can be more than one such message, because $<$ is a partial order). This message M_s has been sent by some S_j. When received, if M_s cannot be delivered we have (see Section 3.2):

$\exists k$ such that $\text{DELIV}_i[k] < \text{ST}_{M_s}[k, i]$.

Let $\text{ST}_{M_s}[k, i] = x$, and consider M_x, the xth message sent from S_k to S_i. From $\text{DELIV}_i[k] < x$ and Proposition 7 we deduce that M_x has not been delivered to S_i. Moreover, by Proposition 1, we have $\text{SEND}(M_x) \rightarrow \text{SEND}(M_s)$, i.e. $M_x < M_s$. This means that M_s is not one of the smallest messages that cannot be delivered to S_i, which shows the contradiction. It follows that every message received by S_i will eventually be delivered.

5. Conclusion

The causal ordering of messages reduces the nondeterminism due to the asynchrony of communication channels. Some examples have been given to illustrate the usefulness of this ordering notion. A simple algorithm implementing this notion, based on message counting, has been proposed. With this implementation, the destination site of a message knows, when receiving a message, if it can be immediately delivered, or if causally preceding messages are still underway. In contrast to the Isis potentially unbounded implementation, the information added to messages to ensure causal ordering is here an $n \times n$ matrix, where n is the number of sites (each entry of the matrix is however not bounded). The same bound is obtained in [15], but the given implementation is more complicated. In contrast, the proposed implementation is simply based on counting the messages emitted. In a certain sense, this implementation extends the concept of sequence numbers used in networks [16]. More generally, the counting techniques are usual in many solutions to synchronization problem, starting from semaphores in centralized systems (to solve competition or synchronization problems), up to counters in distributed systems used to obtain consistent global information (for example, distributed termination [6,10]) or to ensure coherent global decisions (for example, distributed mutual exclusion [13]).

Acknowledgment

The authors would like to thank the referees for their suggestions that helped to improve the clarity of the paper.

References

[1] B. Awerbuch, Complexity of networks synchronization, *J. ACM* **32** (4) (1985) 804–823.

[2] K.A. Barlett, R.A. Scantlebury and P.T. Wilkinson, A note on reliable full duplex transmission over half duplex links, *Comm. ACM* **12** (5) (1969) 260–261.

[3] K. Birman and T. Joseph, Reliable communications in presence of failures, *ACM Trans. Comput. Sci.* **5** (1) (1987) 47–76.

[4] K. Birman and T. Joseph, Exploiting replication in distributed systems, in: S. Mullender, ed., *Distributed Systems* (ACM, New York, 1990).

[5] C. Fidge, Timestamps in message-passing systems that preserve the partial ordering, in: *Proc. 11th Australian Computer Science Conf.*, University of Queensland, 1988.

[6] J.M. Helary, C. Jard, N. Plouzeau and M. Raynal, Detection of stable properties in distributed applications, in: *Proc. 6th ACM Symp. on PODC* (1987) 125–136.

[7] J.M. Helary and M. Raynal, *Synchronization and Control of Distributed Systems and Programs* (Wiley, New York, 1990), p. 200.

[8] T. Joseph and K. Birman, Low Cost Management of Replicated Data in Fault-Tolerant Distributed Systems, *ACM Trans. Comput. Sci.* **4** (1) (1986) 54–70.

[9] L. Lamport, Time, clocks and the ordering of events in a distributed system, *Comm. ACM* **21** (7) (1978) 558–565.

[10] F. Mattern, Algorithms for distributed termination detection, *Distributed Comput.* **2** (3) (1987) 161–175.

[11] F. Mattern, Time and global states of distributed systems, in: *Proc. Internat. Workshop on Parallel and Distributed Algorithms,* Bonas, France, 1988 (North-Holland, Amsterdam) 215–226.

[12] G. Neiger and S. Toueg, Substituting for real-time and common knowledge in distributed systems, in: *Proc. 6th ACM Symp. on PODC* (1987) 281–293.

[13] M. Raynal, *Algorithm for Mutual Exclusion* (MIT Press, Cambridge, MA, 1986), p. 107.

[14] M. Raynal, *Distributed Computation and Networks: Concepts, Tools and Algorithms* (MIT Press, Cambridge, MA, 1988), p. 166.

[15] A. Schiper, J. Eggli and A. Sandoz, A new algorithm to implement causal ordering, in: *Proc. 3rd Internat. Workshop on Distributed Algorithms*, Nice, Lecture Notes in Computer Science 392 (Springer, Berlin, 1989) 219–232.

[16] W. Stenning, A data transfer protocol, *Computer Networks* 1 (1976) 99–110.